DISA 2023

World Symposium on Digital Intelligence for Systems and Machines

PROCEEDINGS

IEEE

September 21-22, 2023
Košice, Slovakia

DISA 2023

World Symposium on Digital Intelligence for Systems and Machines

PROCEEDINGS

September 21-22, 2023
Košice, Slovakia

Website: https://www.disa2023.org/

ISBN: 979-8-3503-4352-6
IEEE Catalog number: CFP23P13-USB
Proceedings editor: Peter Sinčák, Ján Magyar, Martina Szabóová

COMMITTEES

Special Senior Honorary Chair

Shuji Hashimoto, Waseda University, Japan

Special Senior Advisory Board

Takahiro Yamanoi, Hokkaido University, Japan
Xiaoping Chen, University of Science and Technology of China
Hamido Fujita, Iwate Prefectural University, Japan
Vladimír Mařík, Czech Technical University in Prague, Czech Republic
Kaoru Hirota, Beijing Institute of Technology, China
Janusz Kacprzyk, Polish Academy of Sciences, Poland
Imre Rudas, Obuda Univeristy, Hungary
Okyay Kaynak, Bogazici University, Turkey

Special Hosting Honorary Chairs

Stanislav Kmet, Rector, Technical University of Kosice, Slovakia
Gabriel Galgoci, President of AI Slovakia
Liberios Vokorokos, Dean, Faculty of Electrical Engineering and Informatics, Technical University of Kosice, Slovakia
Jozef Zivcak, Dean, Faculty of Mechanical Engineering, Technical University of Kosice, Slovakia

General Executive Chairs

Peter Sincak, Technical University of Kosice, Slovakia
Pitoyo Hartono, Chukyo University, Nagoya, Japan

Review Management Chairs

Anna Bicekova, Technical University of Kosice, Slovakia
Martin Sarnovsky, Technical University of Kosice, Slovakia
Peter Bednar , Technical University of Kosice, Slovakia
Erik Kajati, Technical University of Kosice, Slovakia
Jan Magyar, Technical University of Kosice, Slovakia
Martina Szaboova, Technical University of Kosice, Slovakia

Review Coordination Chairs

Jan Paralic, Technical University of Kosice, Slovakia
Marek Bundzel, Technical University of Kosice, Slovakia
Michal Gregor, University of Zilina, Slovakia
Ivan Virgala, Technical University of Kosice, Slovakia
Laszlo Kovacs, University of Miskolc, Hungary

Ivana Budinska, Slovak Academy of Sciences, Slovakia
Igor Farkas, Comenius University, Bratislava, Slovakia

Plenary and Keynote Management Chairs

Milos Oravec, Slovak Technical University, Bratislava. Slovakia
Matus Pleva, Technical University of Kosice, Slovakia
Peter Papcun, Technical University of Kosice, Slovakia
Ivan Kotuliak, Slovak Technical University, Bratislava. Slovakia
Robert Hudec, University of Zilina, Slovakia

Industry Supporting Organizations

ITAS Slovakia
Digital Coalition Slovakia
Slovak-American Chamber of Commerce, Slovakia

Special Session Chairs

Kristina Machova, Technical University of Kosice, Slovakia
Viera Bordoy, AI Slovakia

Local Organizing Chairs

Stefan Fejedelem, Elfa, Slovakia

Web Management Chairs

Nikola Dobrovska, AI Slovakia

Publicity Chairs

Viera Bordoy, AI Slovakia
Nikola Dobrovska, AI Slovakia

Program Committee

Christopher R. Hudson, Mississippi State University, USA
Peter Papcun , Technical University of Kosice, Slovakia
Daniel Carruth, Mississippi State University, USA
Marian Mach, Technical University of Kosice, Slovakia
Tomohiro Shibata, Kyushu Institute of Technology, Japan
Leszek Rutkowski, Czestochowa University of Technology, Poland
Mária Lucká, Kempelen Institute of Intelligent Technologies, Slovakia
Chihyung Jeon, KAIST, South Korea
Filippo Cavallo, Universita di Firenze, Italy
Jozef Juhar, Technical University of Kosice
Ivana Budinska, Slovak Academy of Sciences, Slovakia

Local Organizing Chairs

Organizing Committee

TABLE OF CONTENTS

Referencing validity assignment using B+tree index enhancements

Michal Kvet
Department of Informatics, Faculty of Management Science and Informatics
University of Žilina
Žilina, Slovakia
Michal.Kvet@uniza.sk

***Abstract*—Temporal databases border each data state by the date and time frame values expressing validity or transaction reference. There are several architectures forming the temporal ecosystem, reflecting the processed precision and granularity. In a general perspective, to get the data image, each data state holds valid or outdated status, expressed by the Boolean value. This assumption is used in this paper. Oracle Database 23c introduced Boolean data type in SQL. This paper deals with temporality modeling, followed by data structure enhancements. It focuses on the performance by introducing B+tree index enhancements, if the data reliability and validity frame assignment cannot be precisely specified, respectively, the reference cannot be adequately set.**

Keywords—Temporal database, Oracle Database 23c, Performance, Validity, Indexing

I. Introduction

Current information systems require significant data to be handled, operated, processed, and stored. Moreover, it is not only about the data that must be treated. It is always necessary to refer to the validity and representation. Thus, intelligent information systems supporting complex decision-making, prognoses definition, and providing analytical outputs require storing not only current valid states but the whole spectrum needs to be stored and evaluated. Therefore, instead of replacing the existing state in case of executing an update operation, the original state is marked invalid, and a new version is loaded, delimited by the current date and time value. To make the system general, the date and time reference can be set, so the state can be either antidated or the new state change can be set to become valid at the specific timestamp in the future. So, there can be plans for the states to become valid later, current valid states, as well as states which were in operation in the past.

Other temporal references can be present except for validity, e.g., specifying transaction reference to enable making data state corrections. Generally, each state may become valid at a different time than this information is actually entered into the database. Consequently, multiple time spectra need to be treated and evaluated to make the system robust.

In this paper, firstly, temporal architectures and data flow perspectives are summarized, highlighting the processed granularity and transfer of historical states into data warehouses forming the analytical interface.

Temporal databases form the main architecture for the defined environment and performance evaluation used in this paper. Even though the states are bordered by the date and time references, many times, it is just essential to compose two data sets for the data image – valid and invalid states. Such a task requires marking each state by the Boolean value at a specified timestamp referenced by the data image. Basically, it can be assumed that each state would be bordered by the valid or invalid state. Unfortunately, in the temporal model, it is, however, usually impossible to achieve this unambiguously [3] [6] [11]. Namely, some state fragments can be only partially valid, either caused by the missing values for the attributes or by violating some integrity rules and constraints, resulting in the necessity to form three-value logic.

In addition to the already mentioned goals, this paper deals with the performance and modeling of the time-delimited states and their reference in the index. Whereas the undefined value cannot be mathematically evaluated and placed in the timeline reference, NULL values are not indexed by requiring the system to make the sequential data scanning, block-by-block, in case of the chance, the NULL value can be present in the result set or by allowing this column integrity rule in the data model definition. We introduce B+tree index enhancements to make the undefined temporal references be placed in the index. Thanks to that, the whole system's performance can significantly rise because of the optimized traverse path and block identification using the index structure.

This paper is structured as follows. Section 2 deals with the data sets and environment characteristics. Section 3 deals with the temporal data model references and granularity perspectives. A subsection takes the slicing process making the data flag of the valid or invalid FIR set image reference. Section 4 deals with and discusses Boolean data types to be used in SQL. In PL/SQL, Boolean data type has been available and verified for many years. However, in SQL, it was introduced in April 2023 in version Oracle Database 23c. Section 5 deals with the B+tree indexing, focusing on the own contribution as the index enhancements. Performance study, processing time, and storage demands are present in section 6.

II. Data set and environment characteristics

For the evaluation, the structure of the used data set was related to the airspace monitoring by taking flight identifier (ECTRL_ID), sequence order reference for the particular flight (SEQ_NUM), positional data, and flight parameters (FLIGHT_DATA) in a binary large object (BLOB) format, assignment to the flight information region (FIR) and entry (TIN) and exit time (TOUT). Please note that the regions for the FIRs also evolve over time, so there can be undefined values present. Moreover, the communication network does not need to be consistently reliable, thus, there can be missing FIR references, as well as data, which were inserted later than the timestamp of force. Consequently, taking any snapshot (image) of the database, undefined values can be present for the positional data and validity, respectively transaction reference, due to delays. To represent the snapshot data and transform temporal sphere into the conventional layer, FIR assignment is taken by the Boolean expressing whether the row is referenced by the particular FIR or not. These snapshots are created at a defined timepoint for the specific FIR by taking the list of flights and a flag, whether the flight is covered by the FIR or not. The source data set included 5 million rows referencing European airspace regions fragments for four years.

The performance evaluation was done in the temporal database environment run on the computer with these parameters: AMD Ryzen 5 PRO 5650U processor with Radeon Graphics, 2.30 GHz, powered by 64 GB RAM (DDR-4, 3200MHz) in total in two slots. The database was stored in 2TB NVMe disc (3500 MB/s for read/write operations). The used version of the database system was Oracle Database 23c Free, Release 23.0.0.0.0 - Developer-Release Version 23.2.0.0.0, run on the Oracle Linux operating system. This type of database and vendor was chosen based on several criteria. Namely, Oracle is the most dynamic evolving type in the research and commercial sphere and provides the best performance. Moreover, autonomous processing in the Cloud environment is offered, so there is no administrator intervention necessity to be made. Additionally, this paper is supported by the Erasmus+ project EverGreen [20], in which Oracle participates as a supporting institution. Finally, Oracle Corporation supports universities through the Oracle for Research project.

III. Temporal Data Models

Temporal databases were introduced almost immediately after the first releases of relational databases. It was initially apparent that conventional systems storing only current valid states would not provide sufficient power and robustness. On one side, data storage was expensive, defective, and too slow for the enormous data set processing. On the other hand, there was a strong demand for temporal extension allowing to store the whole data state evolution and mapping to the timeline [4]. Later, the discs became cheaper, and their capacities were extended. Thanks to that, the hardware part of the problem could be considered as (at least partially) solved. Thus, there was an emphasis on the software and overall architecture of the system. Object-level temporal system comes from the conventional paradigm by extending object identifiers using the temporal references, commonly operated by the validity [10]. Validity is usually defined by the first point (BD) and last timepoint (ED) of the applicability. Besides, transaction references can be present, allowing to make corrections on the temporal systems or to allow users to store states, which were loaded later than the BD of the applicability. If one temporal dimension is used, a uni-temporal system is defined. By referencing two temporal dimensions, bi-temporal architecture is used [11]. Generally, multi-temporal solutions can be deployed by claiming more than two temporal spheres [8] [11].

Object-level temporal architecture is based on the object identifier extension, thus, change on any data attribute automatically requires storing the whole state, consequencing in the necessity to make data change synchronization and grouping attributes based on the frequency of the change. Otherwise, many duplicate column values would be present, extending the storage demands. Storing specific flags for attributes, which were not changed, does not make strong sense [12] because there can be a problem in identifying the physical value. Moreover, getting data images at a defined timepoint would be too time- and resource-consuming [13].

Attribute-oriented granularity enhances each table column by the temporal sphere. Therefore, the image at the defined timepoint is composed by unifying individual attribute values. The main advantage of the attribute-oriented approach is the ability to have any set for the change operation. Physically, the change itself is registered in the temporal layer making the ability to compose the state. No duplicates are present. Moreover, the processing can directly point to the subset of attributes instead of getting the whole state. On the other hand, the state construction can also be demanding if the table is defined by too many attributes. Finally, if the whole state (all attributes) is updated in the first phase, the processing must be divided into individual attributes, which are then processed separately. That is, the already constructed state is destroyed and broken down into individual attributes.

The inter-solution between attribute and object granularity is formed by the group-level temporal attributes [3] [12]. Using the machine-learning techniques and prediction, synchronization groups are detected, formed, and released dynamically. Thus, instead of referencing each data attribute separately, the whole group can be used as a unit. Fig. 1 shows the architecture of the group-level temporal system. Compared to the attribute-oriented granularity, there are additional layers forming, managing, and releasing temporal groups. Temporal groups can be formed by individual attributes, or existing groups can be used, as well. Each group has validity borders, and cascade option is used. Thus, if any element part of the group becomes invalid, the whole group automatically becomes invalid and cannot be used later on.

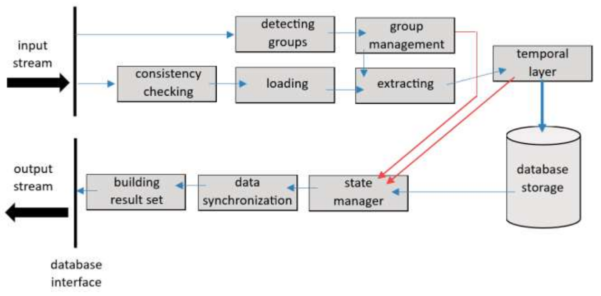

Fig. 1. Group temporal data modeling

A. *Slicing data using boolean data values*

Dealing with the airspace positioning data, it is necessary to build a snapshot of the system at defined timestamp. The snapshot image consists of the whole state, all airspace positions, FIR assignments, flight parameters, like destination, height, speed, restriction conditions, weather conditions, etc. Fig. 2 shows the process of the snapshot building. It is formed as a conventional system with time occurrence reference. Besides, it is possible to get the snapshot image valid during the specified period, either delimited by the full or partial option, expressing the assignment to the FIR region. Namely, if the full option is used, the aircraft must be in the defined region during the whole time period. Otherwise, the FIR assignment is marked as false. Vice versa, if the partial option is used, at least one time point should be covered by the FIR assignment.

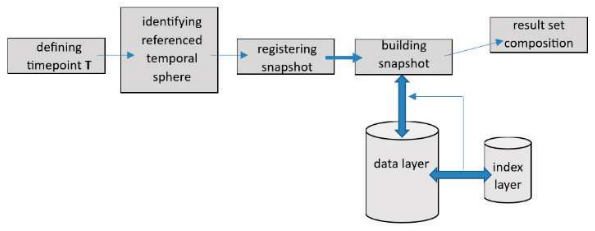

Fig. 2. Slicing – snapshot management

As evident when dealing with the snapshots, there is no primary temporal reference holding the entry and exit time for the particular FIR. Instead, there is a bi-valent variable. TRUE means that the aircraft is part of the FIR at a defined time stamp. FALSE reflects no assignment for a specific FIR. Thus, the snapshot takes the list of the FIRs for each flight by making an assignment bitmap [14] [16]. The schema of the data snapshot is in fig. 3. Each object is represented by a set of states (shown as a rectangle). At the defined time point, it can happen that the object is not covered by a valid state.

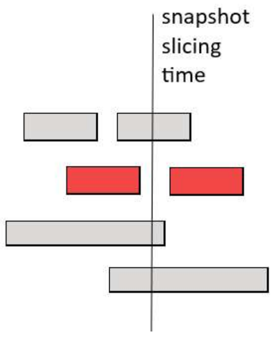

Fig. 3. Sliced data image

IV. Introducing Boolean data type in SQL

In this paper, Oracle Database is used for the processing and evaluation. The Cloud infrastructure, autonomous transaction database, and autonomous warehouses are commonly used in the area of flight monitoring [12]. Moreover, Oracle Database is the most progressively evolving by implementing various enhancements to ensure performance, robustness, and scalability [1] [2] [15]. In addition, autonomous database types limit the administration necessity because it manages itself, and individual administrator processes (patching, updating, expanding, etc.) are left to the cloud vendor [1] [2].

To compose data airspace snapshot, assignment to the FIR is expressed by the bi-valent value. However, Oracle Database did not offer Boolean data type. Instead, it was modeled either by the numerical representation [18] expressed by 0 and 1 or by textual representation delimited by the full string [19] – TRUE / FALSE or by one character [19] holding Y, T, or 1 for TRUE or N, F, 0 for FALSE.

Oracle 23c version released in April 2023 introduced Boolean data type available in SQL and data modeling. Data attributes can now be enhanced directly by the Boolean data type. The performance of the transformation and mapping principles can be found in [12] [18 [19]. Generally, positive (TRUE) value can be expressed directly by the TRUE or true in lowercase, but also textual representations or numerical definitions can be used:

- Numerical – 1/0
- Textual - '1'/'0', 'true'/'false', 'TRUE'/'FALSE', 'on'/'off', 'ON'/'OFF', 't'/'f', 'T'/'F', 'y'/'n', 'Y'/'N'
- Boolean – true/false, TRUE/FALSE

The display format of the Boolean data type value depends on the drivers being used. SQLcl up to 23.1 version will display Boolean values as 1 and 0, while newer versions recognize them as true/false.

The limitation of the Boolean data type usage is related to the data reliability. Namely, FIR regions are not static, and their borders can evolve over time. Moreover, flight data do not need to be precisely accurate, caused by communication network failures, delays, or data capacity. All these factors can lead to process and store undefined values. NULL values are an inseparable part of the Boolean data value processing by getting three-valued logic. Thus, the table attribute can generally hold TRUE/FALSE, but also NULL value. The principles of the three-valued logic and provided function results can be found in [11]. The limitation of using three-valued logic is the search process and related database system performance, which is commonly enhanced by the indexes. The next section deals with the database indexes by propagating proposed enhancements to reach the complexity and performance of the system.

V. Indexing enhancements

Transaction database types can be characterized by the B+tree indexes, formed by the root node as a starting position for the traversing, internal nodes, and leaf layer consisting of the addresses of the tuples in the physical database storage. By using the ROWID address as a logical pointer, data rows can be easily located. It takes 10-bytes – object identifier, referenced data file, data block, and position of the row in the block. The limitation of the ROWID is related to its accuracy. If the position in the physical data layer is to be changed for the row, ROWIDs are not updated. Instead, the original block is enhanced by the pointer locating the correct data block where the row resides. Such a situation is called the row migration process and is caused by the insufficient size to serve the row after the update operation. Data migration is one of the strongest limitations of the index access performance because multiple data blocks need to be memory loaded to locate the row. Theoretically, one block can be migrated multiple times, so the overall performance can significantly degrade [11].

Another problem relates to the chaining if the row size exceeds the capacity of one block. This limitation can be solved only by reconstructing the whole database by changing the processed block size [11]. However, note that the block size is a constant parameter characterizing the database. It defines the structure of the data files in the database layer, but also the memory matrix structure – Buffer cache, to which the blocks are loaded for processing and evaluation. Additionally, block size influences many other supporting structures and loading processes, as well. So, by attempting to change the block size, the whole reconstruction of the database, as well as instance memory, would be required.

Other types of indexes used in transactional systems include the Hash index, which cannot be directly specified in Oracle Database using a keyword but can be emulated using a function-based index [5] [12]. The reason for refusing Hash index definition in Oracle Database is associated in the inability to build a function which is scalable and robust among the huge data source, structure, and amount changes.

Data warehouses, lakes, and other analytical-oriented structures can benefit from bitmap index definition, which is assignment-oriented using Boolean type [18]. However, it can be effectively used only for tuples with a strict definition of the domain values, whereas the bitmap structure must be fixed and predefined in advance [19]. Otherwise, bitmap reconstruction would be necessary, limiting the performance. Moreover, analytical databases are query oriented and do not focus on the data source change operation performance [17]. Thus, if new FIR is created or borders are changed, the whole bitmap structure needs to be rebuilt. Besides, pure bitmap indexes cannot cover temporal elements and validity reflection, so they cannot be used for temporality modeling.

Therefore, it is necessary to develop another solution that can provide robustness and be performance-effective.

A. *Proposed solution*

Whereas the airspace, region, and flight monitoring systems are dynamic, consisting of the sensor-based network providing the data, the core part of the index strategy is based on B+tree index. It has, however, two significant drawbacks. Firstly, B+tree requires high selectivity – the ratio between the number of unique rows to the total number of tuples. While the row is identified by the FIR, performance assessment is clear, while the number of FIRs is limited, and each flight can be decomposed by the FIR assignment matrix regularly during the defined slicing period. Secondly, inaccuracies may arise in the processing process resulting in storing undefined values for the assignment. NULL values, however, cannot be part of the index because such values cannot be mathematically compared and evaluated, so the index traversing process would be impossible to be done. On the other hand, even bitmap index is not suitable due to high processing demands in case of change operation, either in the row, but also in columnar format.

To ensure the performance and serve the dynamic workload, proposed architecture is based on multi-index usage. Several architectures and enhancements were used aiming to find the best suitable solution. Individual types will be sequentially described and evaluated. All of the enhancements originate from the following architecture formed by multiple layers:

The first layer consists of the flight references using B+tree index. The key is the flight_id reference. However, the leaf layer does not deal with the ROWID because it would require the database manager to load the whole block for evaluation and FIR assignment detection. Consequently, for the specified flight, the whole object definition would be necessary to be loaded, which is physically formed by multiple rows, as the data are periodically collected. Therefore, the leaf layer of the first layer consists of the reference to the internal B+tree (the second layer), formed for each flight dynamically. That structure is referred by the flight identifier. The index key is the time occurrence. The leaf layer of the internal B+tree holds the temporal position (modeled commonly by the validity as uni-temporal architecture) and the reference to the bitmap.

The third layer of the proposed multi-index architecture is formed by the bitmaps, which express the assignment to the FIR regions using Booleans.

The proposed architecture is shown in fig. 4. It consists of three layers:

- object identification layer using B+tree (based on the flight_id reference),
- temporal reference (expressing validity frame) in B+tree format,
- FIR assignment using bitmap mapping.

Please note that the bitmap layer always gets one row for each temporal reference. Thanks to this, it is possible to apply dynamic boundary changes of the FIR regions. Whereas the region definitions are also temporal oriented, references used in the second layer can directly apply the correlation. Although it requires some overhead for the bitmap definition, by attempting to make a general solution, significant processing time extensions would be necessary in case of requesting the change of the boundaries at a defined temporal moment. Additionally, it would mean the need to recalculate the entire structure.

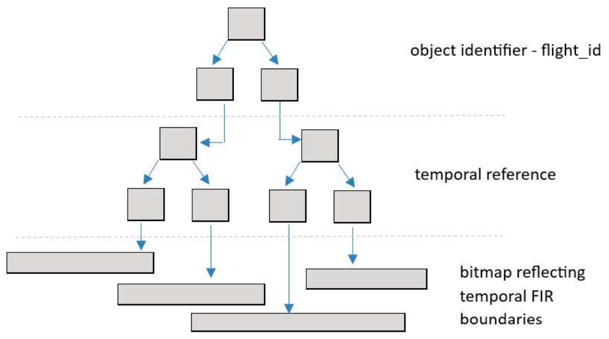
object identifier - flight_id
temporal reference
bitmap reflecting temporal FIR boundaries

Fig. 4. Proposed base architecture – index enhancement using temporal reference and bitmap rows

B. *Enhancements using bitmap priority*

In the preceding core solution, the third layer consists of the bitmap row for each leaf node of the temporal reference layer. One aircraft can be assigned to only one FIR at anytime. Thus, the bitmap must be scanned from the beginning to the occurrence of the value 1 for the bitmap element. Technically, it can cause the whole row consists of zeros only. In that case, data were improperly defined or temporal delays were present. To accelerate the searching process, three enhancements were implemented by introducing the priority in the bitmap composition. The first solution in this category applies the size of the FIR region as a priority. It is based on the assumption that a larger area of the region can cover more flights. However, individual flights usually use predefined trajectory and optimization techniques to limit the costs. Moreover, some regions are restricted or not preferred to be used (e.g. due to the war, sanctions, etc.), as evident in fig. 5. There rules can evolve over time, as well.

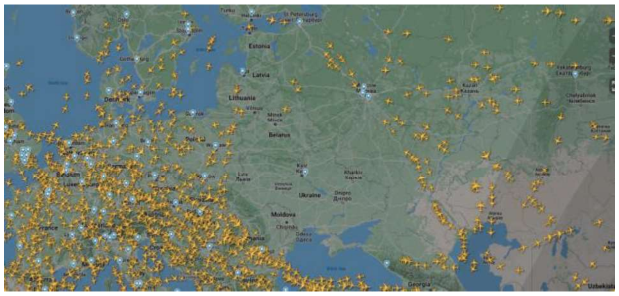

Fig. 5. Flight positions across the European region

Therefore, the second proposed solution emphasizes the usability of the region, instead of its size, as a priority. It provides better performance.

The third solution reflects the neighborhood of the FIRs. It extends the original architecture by the neighborhood stitching reflecting the previous position. The priority is based on the assignment of the region in the recent past by raising the priority of the surrounding regions. The architecture of the system reflecting the stitching and bitmap reference order is shown in fig. 6.

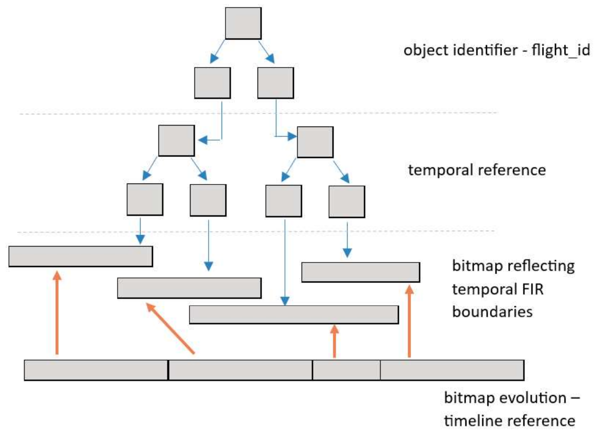
object identifier - flight_id
temporal reference
bitmap reflecting temporal FIR boundaries
bitmap evolution – timeline reference

Fig. 6. Architecture enhanced by the Bitmap evolution layer

C. *Enhancement using multi-temporal system*

The second layer of the proposed solution is expressed by the temporal reference using one dimension. To make the system generally applicable in any temporal sphere and multi-temporal solution, it is necessary to make the system universal respecting multiple dimensions. The temporal reference layer can be preceded by the temporal sphere extractions, followed by using multiple B+tree indexes, requiring adding synchronization layer mapping temporality. The architecture overview is shown in fig. 7.

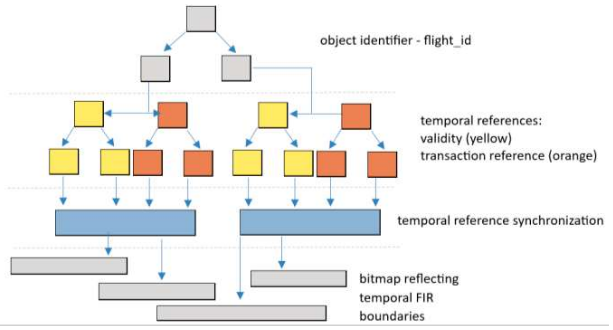
object identifier - flight_id
temporal references:
validity (yellow)
transaction reference (orange)
temporal reference synchronization
bitmap reflecting temporal FIR boundaries

Fig. 7. Synchronization across temporal spheres

In the above case, multiple temporal elements can be present. To build the result set, synchronization across temporality spheres must be present, which can form the bottleneck of the system, even if we do not need to consider all temporal spheres. The second proposed solution in this category is based on temporality cascading (fig. 8). Thus, the leaf layer of the temporal reference takes two pointers – an original bitmap reference and a pointer to the less-level temporal sphere. It can be considered a reference to states regarding their top-level validity. Individual state corrections are stored in the second (lower) level but still in the temporal reference layer.

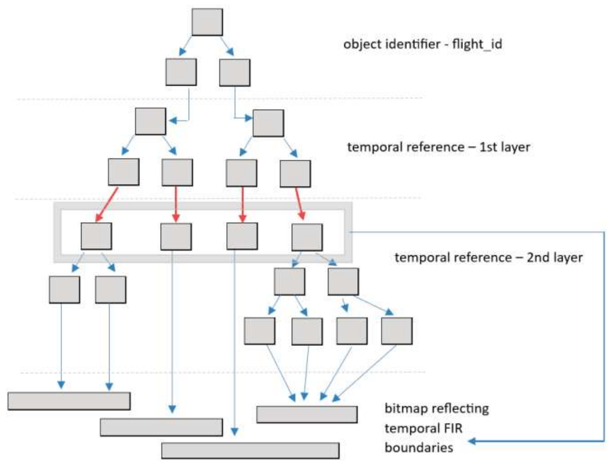
object identifier - flight_id
temporal reference – 1st layer
temporal reference – 2nd layer
bitmap reflecting temporal FIR boundaries

Fig. 8. Temporality cascading

The last processed solution uses the same level for all temporal spheres, but for each state, temporal coverage is defined, pointing to the timeline. Thanks to that, all spheres can be easily identified, processed, and evaluated.

D. *Enhancement using Hash Balancer*

B+tree structure is an always balanced structure, which maintains efficiency regardless of the increase in the amount of data. The balancing process ensures that the traverse path height is the same to access any leaf node from the root element. Thanks to that, query processing can significantly benefit. On the other hand, any change operation (insert, update, or delete) requires the index to be balanced before the operation is done and the transaction approved. Thus, rebalancing can be considered a major limiting factor in index performance when changing data states. In addition, this is significantly reflected in the temporal layer, where the key to the index is the time element. The time values are constantly growing and are therefore included in the right part of the index, which tends to degrade into a linear linked list. As a result, with any change, it is necessary to perform index rotations to ensure it is balanced. Therefore, another enhancement of the proposed solution adds a preprocessing layer to the temporal element processing. Instead of storing direct values, they are hashed to be randomly distributed in the index, so the rebalancing demands are lowered. On the other hand, it is necessary to hash the value before indexing itself.

To serve the solution and ensure performance for the change operations, one extra strategy has been discussed. The goal is to ensure the even loading of individual branches of the temporal layer and indexes in it and thus limit the need for constant balancing. Thus, instead of indexing temporal value itself (delimited by the timestamp consisting of the date and time elements up to the defined second fraction precision), only time elements are considered as the index keys. Based on the workload, hours, minutes, or seconds are taken. Namely, for the massive traffic, the combination of the elements is time preferred. On the contrary, if the region is not used constantly throughout the day, the use of minute and second elements exclusively is preferred (refusing indexing hours). For example, if the region was not used at night, the system would degrade from the point of view of the distribution of values in the hourly spectrum.

E. Indexing data snapshots

For route planning, analytics, and taking future plans, it is necessary to produce data snapshots periodically. Using the data snapshot, individual analytic-oriented operations are done. Thus, the emphasis should be done not only for the original data source but also data snapshots, and sliced data access should be optimized. Boolean values are there taken into consideration, even enhanced by the NULL values. The existing approaches limiting NULL values by transforming them into another applicable value can be implemented only for the numerical or textual representations of the bivalent values. The function-based index [4] [11] or virtual column value calculation [4] [11] brings additional demands by storing the references physically in the index.

Our proposed solution is based on B+tree extension using dynamic transformation, operated by the introduced Value Separator and Indexer background process. The overall architecture is shown in fig. 9. By introducing this extension, even NULL values can be located and referred by the index. Namely, three indexes are created, separated for each value – TRUE, FALSE, and NULL. These three indexes are more compact and the balancing process can be done in parallel for each processed element. The key for the index is the unique row identifier – flight_id.

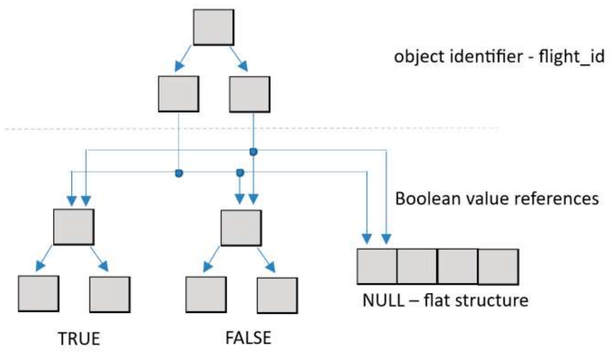

Fig. 9. Architecture using three indexes for the Boolean representation

By attempting to group all indexes into a common structure, worse performance will be reached. Namely, firstly, the balancing process would be more demanding as the structure is bigger. Secondly, whereas each FIR can hold only one value, just a relevant structure needs to be updated and balanced instead of the whole structure. And thirdly, it would always be better to make the neighborhood enabling for the processing. If the trajectory of the airplane is available, then the consecutive FIR assignment can be predicted (fig. 10). Thus, instead of the whole structure scanning, consisting of the TRUE, FALSE, and NULL values, the priority of the search can be defined to lower the demands. Besides, if three indexes are present, they can be scanned in parallel.

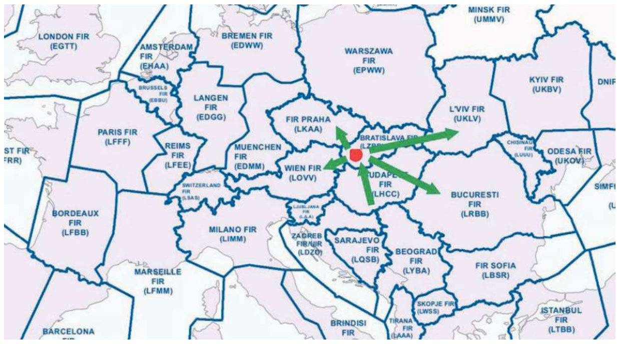

Fig. 10. FIR neighborhood

VI. PERFORMANCE STUDY

In this paper, the slicing process and data reconstruction by the extended support of the analytics, decision making, and prognoses evaluation were proposed. The assignment of the FIR can be defined either by the temporal references or by Boolean values in the snapshots. This section aims at performance evaluation of the proposed approaches, compared to the existing solutions, to declare the efficiency but also identify limitations. Whereas the processing is dynamic and depends markedly on the data set structure, during the evaluation, three data sets were combined, differing in the holding amount of tuples. The primary aim is to ensure global performance, but also to make the demands as linear as possible for the scalability operations.

The data set consisted of 5 million rows.

During the evaluation, three parameters were studied – the size of the whole index structure covering all the auxiliary structures, select statement processing time to obtain the current state for each flight in a snapshot by referring only FIRs, which are covered by a specific airplane at the defined timestamp, not the whole list of FIRs. The third evaluated parameter declares the usability in case of requesting new data to be loaded by emphasizing the balancing and index update operations.

For the evaluation, seven solutions were used:

- Solution 1 (**S1**) is based on the original B+tree architecture, which cannot hold undefined values.
- Solution 2 (**S2**) transforms undefined values using the function to provide a function-based index, allowing to cover all the data in it.
- Solution 3 (**S3**) – base architecture using validity temporal reference and bitmap boundaries.
- Solution 4 (**S4**) – multiple indexes using bitmap priority.
- Solution 5 (**S5**) – synchronization across temporal spheres. **S5a** uses two temporal spheres, **S5b** uses three temporal spheres, and **S5c** uses four temporal spheres.
- Solution 6 (**S6**) – temporal cascading, **S6a** takes two dimensions, **S6b** uses three dimensions, and **S6c** manages four temporal dimensions.
- Solution 7 (**S7**) – proposed B+tree index extended by the NULL reference layer, operated by the flat table (**S7a**), sorted array based on the object (**S7b**), sorted array based on the states in a timeline (**S7c**). Undefined state reference defined by the B+tree data structure

defined by the state identifier as a key is referred by the solution **S7d**.

Solutions **S1** and **S2** are used as a reference dealing with the undefined values.

A. Size of the structure

In this category, storage demands were evaluated. The left boundary is defined by the S1, by which, however, undefined values cannot be referenced, and thus sequential block scanning is required. Transformation of the undefined values using a function-based index provides the worst solution. The reason is based on the necessity to transform not only undefined values as a reference, but the whole data type for storing FIR assignment was necessary to be rebuilt, while the Boolean value cannot serve transformed undefined values, which can be then indexed.

Based on the reached results, it is evident that the data structure for the B+tree index enhancement (S7) does not make a significant difference in terms of the storage demands – approximately 0.1%. The robust architecture using multiple levels requires additional storage. Compared to the S1 as a reference, it takes an additional 10.9% for S3 up to 25.8% for S6. The structure and layer extension is almost linear in terms of the storage capacity range. Tab. 1 shows the results.

TABLE I. STORAGE DEMANDS

		Storage demands [GB]
S1		4 634
S2		5 920
S3		5 140
S4		5 160
S5	**S5a**	5 210
	S5b	5 380
	S5c	5 820
S6	**S6a**	5 215
	S6b	5 386
	S6c	5 831
S7	**S7a**	4 685
	S7b	4 692
	S7c	4 700

The graphical representation in terms of the column chart is shown in fig. 11.

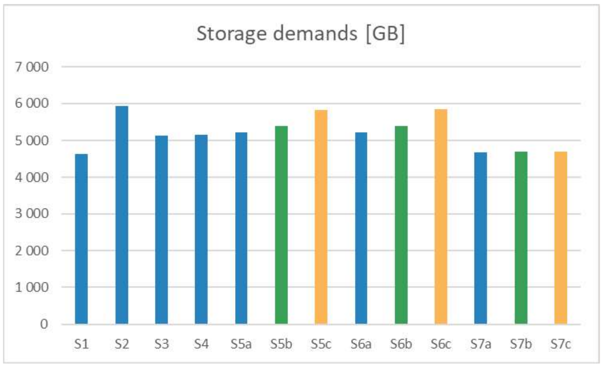

Fig. 11. Synchronization across temporal spheres

B. Processing time demands – getting snapshot

This section emphasizes the processing time demands to get the snapshot. It is evaluated in two streams – getting current data image obtained by the slicing process to get the snapshot (1st stream) and getting image valid in the past – 1 month ago, respectively one year ago (2nd stream). For clarity, the values are expressed in percentage to declare the additional demands. Tab. 2 shows the results. Based on the reached results, it is evident that the slicing is not impacted by the date and time position, whereas the temporal sphere reference is always adequately covered by the architecture. However, this cannot be said about the pure B+tree index (S1), where time dependence is evident. The older the value we need to obtain, the greater amount of data that needs to be processed and evaluated, consequencing in the growth of the processing time. The solutions S1 and S2 create the upper limits for the performance. Moreover, S1 is not scalable by the timeline. The flagged reference value is covered by solution S1.

TABLE II. PROCESSING TIME – DATA RETRIEVAL

		Current snapshot	1 month ago	1 year ago
S1		100	160	220
S2		100	140	176
S3		77	81	85
S4		74	74	74
S5	**S5a**	68	70	73
	S5b	69	70	74
	S5c	72	74	79
S6	**S6a**	71	76	84
	S6b	73	78	89
	S6c	74	82	94
S7	**S7a**	51	51	53
	S7b	48	48	49
	S7c	44	45	46

Based on the Tab. 2 shown above, temporal cascading (S6) is not recommended, while the structure can evolve dynamically using multiple temporal spheres. Unlike temporal cascading, synchronization across the temporal spheres can be hugely done by parallel processing, thus, it is not severely impacted by the referred date and time value of the snapshot. Graphical representation is depicted by fig. 12.

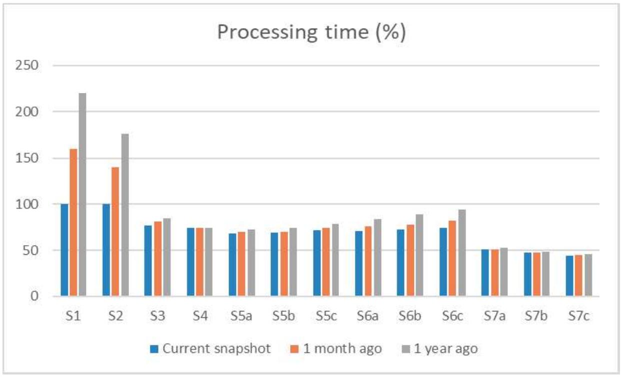

Fig. 12. Processing time – data retrieval

In this step, the processing time reflects the data retrieval by getting the data snapshot at the defined timepoint.

C. *Processing time demands – data loading*

Finally, the computational study focuses on data loading. Evidently, the proposed architecture brings significant performance benefits for the processing time, as well as the storage demands, compared to the function-based solution. However, now let us also reference the data loading process to declare the practical usability of the solution. While there is continuous data streaming considering the flights, it is critical to lower the processing time demands as much as possible.

During the evaluation study, the same solutions were used. The individual proposed solutions are mostly based on the B+tree indexes on multiple levels, so the balancing process is an inseparable part. Tab. 3 shows the results. As always, solutions S1 and S2 are used as reference to point to the performance demands and characteristics.

Transformation using the function elaborates an additional 9.2%. However, the proposed architectures do not rise the processing time demands significantly. Namely, base architecture requires an additional 1.3%. By handling priority, it raises up to 2.1%, reflecting only 0.8% (comparing S4 and S3). The architecture of the synchronization processes across the temporal spheres is worth handling properly. Cascading temporality requires multiple spheres to be treated. However, it cannot be done in parallel. Moreover, the balancing must also be done sequentially from the top level to the bottom. Overally, it takes 6.2% up to 12.9%, depending on the number of temporal spheres. The best solution is done by the index extension handler, by which the original data layer composition remains, but B+tree index is enhanced by the pointers to the undefined values in the state set. The performance depends on the size and demands for the balancing, ranging from 0.8% up to 2.2%.

TABLE III. PROCESSING TIME – DATA LOADING

		Data loading
S1		100.0
S2		109.2
S3		101.3
S4		102.1
S5	S5a	101.5
	S5b	101.9
	S5c	104.3
S6	S6a	106.2
	S6b	108.6
	S6c	112.9
S7	S7a	100.8
	S7b	101.4
	S7c	102.2

When dealing with the performance, it is worth to refer to the scalability and performance impacts regarding the increase in the data set size. The evaluation also dealt with this aspect, however, the proportional characteristics were always identified, thus it can be clearly declared, that the proposed solutions and techniques are scalable. For the clarity, those data set characteristics were used: The smallest data set consisted of 5 million of rows. The medium data set is formed by the 500 million of rows. The structure remains the same. The largest data set applies partitioning techniques in a horizontal manner. Overally, it consists of 50 billion rows, which are, however, split into 100 partitions, defined by the FIR assignment. Fig. 13 shows the results, also declared in the percentage expressing processing time to get the current image, while fig. 14 emphasizes the data loading process.

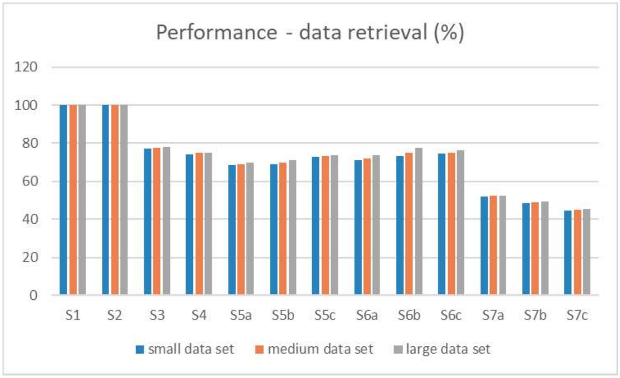

Fig. 13. Processing time – data retrieval – multiple data sets

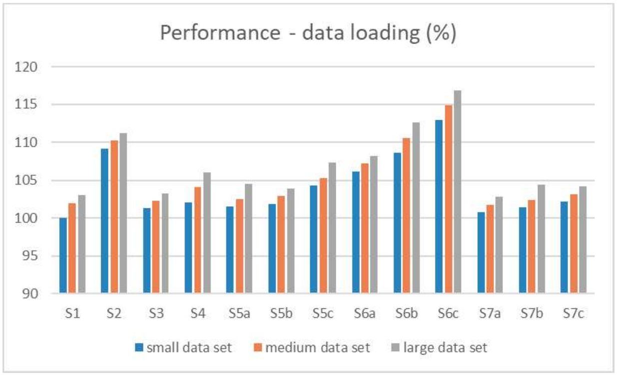

Fig. 14. Processing time – data loading – multiple data sets

VII. CONCLUSIONS

Efficient data management and processing is an inevitable part of information technology. It serves the data storage and individual operations on the database to provide the data in a proper format as soon as possible. This requires various optimization techniques to ensure that. One of the most significant aspects relates to data modeling and indexing.

Although, in this paper, the performance analysis of the flight data streaming is made, the solutions and techniques can be applied to any dynamic monitoring system. The architecture is based on the temporal data, which are sliced at the defined timestamp to get the data image and snapshot.

In this environment, performance optimization was done. It is based on the newly introduced Boolean data type in Oracle Database, which, however, cannot manage undefined values, while indexing. To serve the workload and cover the data inside the indexes, the B+tree index extension is introduced, delimited by the various architectures for undefined value references. Besides, the data slicing process is taken into emphasis, formed by the three-level architecture with various enhancements, if multiple temporal spheres are present. During the performance evaluation study, data retrieval process, as well as data loading, were pointed, preceded by the total size of the structure. Moreover, the evaluation strategy focused on scalability. It can be clearly declared that the proposed solutions provide sufficient power and are prone of changes the amount of data rapidly.

During future research, we will focus on placing the whole architecture in the dynamically partitioned system by making local indexes for each partition but encapsulated by the synchronization layer, if the data need to be moved from one partition to another. Besides, the emphasis will be on dynamic data balancing across the partitions.

ACKNOWLEDGMENT

It was supported by the Erasmus+ project: Project number: 2022-1-SK01-KA220-HED-000089149, Project title: Including EVERyone in GREEN Data Analysis (EVERGREEN) funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the Slovak Academic Association for International Cooperation (SAAIC). Neither the European Union nor SAAIC can be held responsible for them.

Co-funded by the European Union

EverGreen DATA Analytics

REFERENCES

[1] Abhinivesh, A., Mahajan, N.: The Cloud DBA-Oracle, Apress, 2017

[2] Anders, L.: Cloud computing basics, Apress, 2021

[3] Cunningham, T.: Sharing and Generating Privacy-Preserving Spatio-Temporal Data Using Real-World Knowledge, 23rd IEEE International Conference on Mobile Data Management, Cyprus, 2022.

[4] Greenwald, R., Stackowiak R., and Stern, J.: *Oracle Essentials: Oracle Database 12c*, O'Reilly Media, 2013.

[5] Idreos, S., Manegold S., and Graefe, G.: Adaptive indexing in modern database. In: ACM International Conference Proceeding Series, 2012

[6] Jakóbczyk, M.: Practical Oracle Cloud Infrastructure: Infrastructure as a Service, Autonomous Database, Managed Kubernetes, and Serverless, Apress, 2020

[7] Janáček, J. and Kvet, M.: Shrinking fence search strategy for p-location problems, 2020 IEEE 20th International Symposium on Computational Intelligence and Informatics (CINTI), Hungary, 2020

[8] Kuhn, D. and Kyte, T.: *Oracle Database Transactions and Locking Revealed: Building High Performance Through Concurrency*, Apress, 2020.

[9] Kumar, Y., Basha, N., et al.: Oracle High Availability, Disaster Recovery, and Cloud Services: Explore RAC, Data Guard, and Cloud Technology, Apress, 2019

[10] Kvet, M.: Developing Robust Date and Time Oriented Applications in Oracle Cloud: A comprehensive guide to efficient date and time management in Oracle Cloud, Packt Publishing, 2023, ISBN: 978-1804611869

[11] Kuhn, D. and Kyte, T.: *Expert Oracle Database Architecture: Techniques and Solutions for High Performance and Productivity.* Apress, 2021.

[12] Kvet, M., Papán, J.: The Complexity of the Data Retrieval Process Using the Proposed Index Extension, IEEE Access, vol. 10, 2022.

[13] Lewis, J.: *Cost-Based Oracle Fundamentals*, Apress, 2005.

[14] Liu, Z., Zheng Z., Hou, Y. and Ji, B.: Towards Optimal Tradeoff Between Data Freshness and Update Cost in Information-update Systems, 2022 International Conference on Computer Communications and Networks (ICCCN), USA, 2022.

[15] Steingartner W., Eged, J., Radakovic, D., Novitzka V.: Some innovations of teaching the course on Data structures and algorithms, In 15th International Scientific Conference on Informatics, 2019.

[16] Su S.Y.W., Hyun S.J. and Chen, H.M.: Temporal association algebra: a mathematical foundation for processing object-oriented temporal databases, IEEE Transactions on Knowledge and Data Engineering, vol. 4, issue 3, 1998.

[17] Yao, X., Li, J., Tao, Y. and Ji, S.: Relational Database Query Optimization Strategy Based on Industrial Internet Situation Awareness System, 7th International Conference on Computer and Communication Systems (ICCCS), China, 2022.

[18] https://oracle-base.com/articles/23c/Boolean-data-type-23c

[19] https://docs.oracle.com/en/database/oracle/oracle-database/23/sqlrf/Data-Types.html#GUID-A3C0D836-BADB-44E5-A5D4-265BA5968483

[20] Erasmus+ project EverGreen dealing with the complex data analytics: https://evergreen.uniza.sk/